Machine Learning (2021) 110:1429–1462
https://doi.org/10.1007/s10994-021-05992-x

Automated adaptation strategies for stream learning

Rashid Bakirov[1] · Damien Fay[2] · Bogdan Gabrys[3]

Received: 7 October 2018 / Revised: 27 April 2021 / Accepted: 5 May 2021 /
Published online: 2 June 2021

Abstract
Automation of machine learning model development is increasingly becoming an established research area. While automated model selection and automated data pre-processing have been studied in depth, there is, however, a gap concerning automated model adaptation strategies when multiple strategies are available. Manually developing an adaptation strategy can be time consuming and costly. In this paper we address this issue by proposing the use of flexible adaptive mechanism deployment for automated development of adaptation strategies. Experimental results after using the proposed strategies with five adaptive algorithms on 36 datasets confirm their viability. These strategies achieve better or comparable performance to the custom adaptation strategies and the repeated deployment of any single adaptive mechanism.

Keywords Adaptive machine learning · Streaming data · Non-stationary data · Concept drift · Automated machine learning

1 Introduction

Automated model selection has long been studied (Wasserman, 2000) and recently, notable advances in practical automated machine learning (AutoML) approaches (Hutter et al., 2011; Lloyd et al., 2014; Kotthoff et al., 2017; Mohr et al., 2018; Martin Salvador et al., 2019; Olson & Moore, 2019; Kedziora et al., 2020) have been made. In addition, automated data pre-processing in the context of complex machine learning pipelines generation and validation has also been a topic of recent interest (Feurer et al., 2015; Martin Salvador et al., 2019; Nguyen et al., 2020). There is however a gap concerning automated development of models' adaptation strategy, which is addressed in this paper. Here we define *adaptation* as changes in model training set, parameters and

Editor: Joao Gama.

Rashid Bakirov
rbakirov@bournemouth.ac.uk

[1] Department of Computing and Informatics, Bournemouth University, Poole, UK

[2] INFOR/Logicblox, Atlanta, GA, USA

[3] Advanced Analytics Institute, University of Technology Sydney, Ultimo, Australia

structure, all designed to track changes in the underlying data generating process over time. This contrasts with model selection which focuses on parameter estimation and the family to sample the model from.

With the current advances in data storage, database and data transmission technologies, learning on streaming data has become a critical part of many processes. Many models which are used to make predictions on streaming data are static, in the sense that they do not learn on current data and hence remain unchanged. However, there exists a class of models, stream learning models, which are capable of adding observations from the data stream to their training sets. In spite of the fact that these models utilise the data as it arrives, there can still arise situations where the underlying assumptions of the model no longer hold. We call such settings dynamic environments, where changes in data distribution (Zliobaite, 2011), change in features' relevance (Fern & Givan, 2000), non-symmetrical noise levels (Schmidt & Lipson, 2007) are common. These phenomena are sometimes called *concept drift*. It has been shown that many changes in the environment which are no longer being reflected in the model contribute to the deterioration of model's accuracy over time (Schlimmer & Granger, 1986; Street & Kim, 2001; Klinkenberg, 2004; Kolter & Maloof, 2007). This requires constant manual retraining and readjustment of the models which is often expensive, time consuming and in some cases impossible—for example when the historical data is not available any more. Various approaches have been proposed to tackle this issue by making the model adapt itself to the possible changes in environment while avoiding its complete retraining. These approaches however are manually designed and the application of automated machine learning to streaming data is scarce, which is the gap we aim to contribute to.

Typically there are several possible ways or *adaptive mechanisms* (AMs) to adapt a given model. A single iteration of adaptation is achieved by deploying one of multiple AMs (including trivial "doing nothing"), which changes the state of the existing model. Thus, during the model's operation, it is adapted by the sequential deployment of various AMs with the arrival of new data. We call the order of this deployment an *adaptation strategy* (AS). While in most of the existing research these adaptation strategies are custom (i.e. algorithm-specific) and are fixed at the design stage of the algorithm, a sequential adaptation framework proposed in our earlier work (Bakirov et al., 2015) enables flexible adaptation strategies without a prescribed AM deployment order. These flexible adaptation strategies, automatically developed according to this framework can be applied to any set of adaptive mechanisms for various machine learning algorithms. This removes the need to design custom adaptive strategies, resulting in automation of adaptation process. In this work we empirically show the viability of the automated adaptation strategies based on cross-validation (Bakirov et al., 2015) with the optional use of retrospective model correction (Bakirov et al., 2016).

We focus on the batch prediction scenario, where data arrives in large segments called batches. This is a common industrial scenario, especially in the chemical, microelectronics and pharmaceutical areas (Cinar et al., 2003). For the experiments we use Simple Adaptive Batch Learning Ensemble (SABLE) (Bakirov et al., 2015) and batch versions of four popular stream learning algorithms—the Dynamic Weighted Majority (DWM) (Kolter & Maloof, 2007), the Paired Learner (PL) (Bach & Maloof, 2010), the Leveraged Bagging (LB) (Bifet et al., 2010b) and BLAST (van Rijn et al., 2015). The use of these five algorithms allows to explore different types of online learning methods; local experts ensemble for regression in SABLE, global experts ensemble for classification in DWM and LB, switching between the two models in PL and the heterogeneous global ensemble in BLAST.

After a large-scale experimentation with 5 regression and 31 classification datasets, the main finding of this work is that in our settings, the proposed automated adaptive strategies show comparable accuracy rates to the custom adaptive strategies and, in many cases, to the repeated deployment of a single "best" AM. Thus, they are feasible to use for adaptation purposes, while saving time and effort spent on designing custom strategies.

The paper follows by presenting the related work on automated machine learning and adaptive mechanisms in Sect. 2. Section 3 presents mathematical formulation of the framework of adaptation with multiple adaptive mechanisms in batch streaming scenario. Section 4 introduces algorithms used for the experimentation, including their inherent adaptive mechanisms and custom adaptation strategies. Experimental methodology, the datasets on which experiments were performed and results are given in Sect. 5. We give our final remarks in Sects. 6 and 7.

2 Related work

This section provides a background for our research. We start with a review of relevant automated machine learning approaches, particularly those which consider streaming data scenario. We follow up with a broad analysis of ML literature from the adaptive mechanisms point of view, where we introduce a simple hierarchy of adaptation. We then discuss how multiple adaptive mechanisms paradigm has been used for automating the design of predictive algorithms.

2.1 Automated machine learning for streaming data

Automated machine learning is an active research area. So far however, it has been mostly applied to static datasets, and there are not many works which consider automation for streaming scenario. Among these, different approaches exist. One of the works, before the most recent wave of AutoML research, can be found in Kadlec and Gabrys (2009) where a general purpose architecture to develop robust, adaptive prediction systems for the autonomous operation in changing environments for streaming data has been proposed. Various instantiations of this architecture followed focusing on challenging problems from the process industry when building adaptive, predictive soft sensors (Kadlec & Gabrys, 2010, 2011; Bakirov et al., 2017).

Taking advantage of the recent wave of research in AutoML, an alternative approach to adaptation to changing environments was proposed in Martín Salvador et al. (2016) where repeated automated deployment of Auto-WEKA for Multi-Component Predictive Systems (MCPS) to learn from new batches of data was used for life-long learning and the adaptation of complex MCPS when applied to changing streaming data from process industries. Celik and Vanschoren (2021) represent a development of this idea with the inclusion of the drift detection and the experimentation using several open source AutoML frameworks. An interesting approach closely tied with the Auto_Sklearn is described in Madrid et al. (2019). Authors propose using the ensemble nature of this framework to deal with streaming data, by adapting the weights of experts and adding new ones.

Some of the other recently proposed relevant methods are primarily focused on hyper-parameter optimisation problems. For example, Veloso et al. (2018) propose hyper-parameter optimization for streaming regression problems using the Nelder–Mead algorithm. In their experiments they optimise the hyper-parameters of one specific regression method.

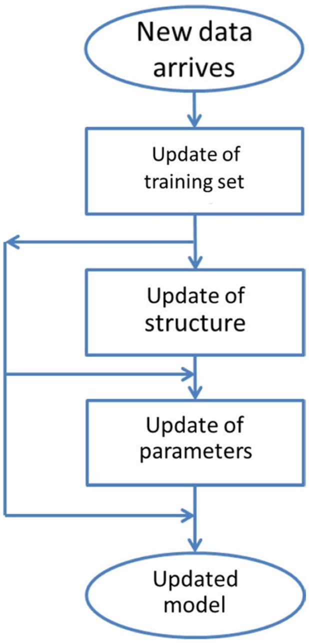


Fig. 1 General adaptation scheme (Bakirov, 2017)

Carnein et al. (2020) are focusing on the hyper-parameter selection for clustering of data in a streaming environment. They propose utilising a dynamic ensemble of different hyper-parameter configurations.

Despite the existing research, as acknowledged and discussed in a recent comprehensive and synthesising review of concepts in AutoML research and beyond (Kedziora et al., 2020), the pursuit of autonomy, described as the AutoML system's capability to independently adapt the ML solution over a lifetime of operation in changing environments, remains a lofty goal.

2.2 Adaptive mechanisms

Adapting machine learning models is an essential strategy for automatically dealing with changes in an underlying data distribution to avoid training a new model manually. Modern machine learning methods typically contain a complex set of elements allowing many possible AMs. This can increase the flexibility of such methods and broaden their applicability to various settings. However, the existence of multiple AMs also increases the decision space with regards to the adaptation choices and parameters, ultimately increasing the complexity of adaptation strategy. A possible hierarchy[1] of AMs is presented in Fig. 1.

In a streaming data setting, to increase the accuracy, it can be beneficial to include recent data in the training set of the predictive models. On the other hand however, retraining a model from scratch is often inefficient, particularly dealing with high throughput

[1] Here, the hierarchy is meant in a sense that the application of an adaptive mechanism of the higher level, requires the application of the adaptive mechanism of lower level.

scenarios or even impossible when the historical data is no longer available. For these cases, the solution is updating the model using only the available recent data. This can be done inherently by some general purpose ML algorithms, e.g. Naive Bayes or using stream/online algorithms, e.g. online Least Squares Estimation (Jang et al., 1997), online boosting and bagging (Oza & Russell, 2001) etc. Additionally, for non-stationary data, it becomes important to not only select a training set of sufficient size but also one which is relevant to the current data. This is often achieved by a moving window (Widmer & Kubat, 1996; Klinkenberg, 2004; Zliobaite & Kuncheva, 2010) or decay approaches (Joe Qin, 1998; Klinkenberg & Joachims, 2000).

The structure of a predictive model is a graph with the set of its components and the connections therein. Some common examples are hierarchical models (e.g. decision trees) or more complex graphs (e.g. Bayesian or neural networks). Here, the structure is not necessarily limited to the topological context—number of rules in rule based systems or number of experts in an ensemble could be considered part of the model's structure. Adaptation can be achieved by updating this structure, for example in decision and model trees (Domingos & Hulten, 2000; Hulten et al., 2001; Ikonomovska et al., 2010), neuro-fuzzy approaches (Gabrys & Bargiela, 1999; Gabrys, 2004; Sahel et al., 2007), neural networks (Carpenter et al., 1991; Vakil-Baghmisheh & Pavešić, 2003; Ba & Frey, 2013), Bayesian networks (Friedman & Goldszmidt, 1997; Alcobé, 2004; Castillo & Gama, 2006) and ensemble methods (Stanley, 2002; Gabrys & Ruta, 2006; Kolter & Maloof, 2007; Hazan & Seshadhri, 2009; Lemke et al., 2009; Ruta et al., 2011; Gomes Soares & Araújo, 2015; Bakirov et al., 2017).

The final layer of adaptation is changing the models' parameters, e.g. experts' combination weights in ensemble methods. These weights are often recalculated or updated throughout a models' runtime (Littlestone & Warmuth, 1994; Kolter & Maloof, 2007; Elwell & Polikar, 2011; Kadlec & Gabrys, 2011; Bakirov et al., 2017). Another group of techniques belonging to this family are methods using meta-learning for model adaptation (Nguyen et al., 2012; Rossi et al., 2014; van Rijn et al., 2015; Lemke & Gabrys, 2010). These methods generally include training a meta-model using meta-features. The meta-model is then used to select one or more predictors to calculate the final prediction. The change of the meta-model can then be seen as the change in parameters of the predictive model.

In this work we consider the possibility of using multiple different adaptive mechanisms, most often at different levels of the hierarchy. Many modern machine learning algorithms for streaming data explicitly include this possibility. A prominent example are the adaptive ensemble methods (Wang et al., 2003; Kolter & Maloof, 2007; Scholz & Klinkenberg, 2007; Bifet et al., 2009; Kadlec & Gabrys, 2010; Elwell & Polikar, 2011; Alippi et al., 2012; Souza & Araújo, 2014; Gomes Soares & Araújo, 2015; Bakirov et al., 2017) which often feature AMs from all three levels of hierarchy—online update of experts, changing experts' combination weights and modification of experts' set. Machine learning methods with multiple AMs are not limited to ensembles, but can also include Bayesian networks (Castillo & Gama, 2006), decision trees (Hulten et al., 2001), model trees (Ikonomovska et al., 2010), champion-challenger schemes (Bach & Maloof, 2010) etc.

2.3 Automating design of algorithms with multiple AMs

Existence of multiple AMs raises questions w.r.t. how they should be deployed. This includes defining the order of deployment and adaptation parameters (e.g. decay factors, expert weight decrease factors, etc.). It should be noted that all of the aforementioned algorithms use custom adaptive strategies, meaning that they deploy AMs in a manner specific to each of them. It follows that designing adaptive machine learning methods is a complex enterprise and is an obstacle to the automation of machine learning model's design. Kadlec and Gabrys (2009) present a plug and play architecture for pre-processing, adaptation and prediction which foresees the possibility of using different adaptation methods in a modular fashion, but does not address the method of AM selection. Bakirov et al. (2015, 2016) have presented several such methods for AM selection for their adaptive algorithm, which are discussed in detail in Sect. 3.2. These methods can be seen as automated adaptive strategies, which are applicable to all adaptive machine learning methods with multiple AMs. This allows simply using the described strategies for model adaptation, once having defined the available AMs.

3 Formulation

As adaptation mechanisms can affect several elements of a model and can depend on performance several time steps back, it is necessary to clarify the concepts via a framework to avoid confusion. We assume that the data is generated by an unknown time varying data generating process which can be formulated as:

$$y_\tau = \psi(\boldsymbol{x}_\tau, \tau) + \epsilon_\tau, \tag{1}$$

where ψ is the unknown function, ϵ_τ a noise term, $\boldsymbol{x}_\tau \in \mathcal{R}^M$ is an input data instance, and y_τ is the observed output at time τ. Then we consider the predictive method at a time τ as a function:

$$\hat{y}_\tau = f_\tau(\boldsymbol{x}_\tau, \Theta_f), \tag{2}$$

where $\hat{y}_\tau$ is the prediction, f_τ is an approximation (i.e. the model) of $\psi(\boldsymbol{x}, \tau)$, and Θ_f is the associated parameter set. Our estimate, f_τ, evolves via adaptation as each batch of data arrives as is now explained.

3.1 Adaptation

In the batch streaming scenario considered in this paper, data arrives in batches with $\tau \in \{\tau_k \cdots \tau_{k+1} - 1\}$, where τ_k is the start time of the kth batch. If n_k is the size of the kth batch, $\tau_{k+1} = \tau_k + n_k$. It then becomes more convenient to index the model by the batch number k, denoting the inputs as $\boldsymbol{X}_k = \boldsymbol{x}_{\tau_k}, \cdots, \boldsymbol{x}_{\tau_{k+1}-1}$ and the outputs as $\boldsymbol{y}_k = y_{\tau_k}, \cdots, y_{\tau_{k+1}-1}$. We examine the case where the prediction function f_k is static within a kth batch.[2]

We denote the a priori predictive function at batch k as f_k^-, and the a posteriori predictive function, i.e. the adapted function given the observed output, as f_k^+. An *adaptive mechanism*, $g(\cdot)$, may thus formally be defined as an operator which generates an updated prediction function based on the batch $\boldsymbol{\mathcal{V}}_k = \{\boldsymbol{X}_k, \boldsymbol{y}_k\}$ and other optional inputs. This can be written as:

[2] A batch typically represents a meaningful real-world segmentation of the data, for example a plant run and so our adaptation attempts to track run to run changes in the process.

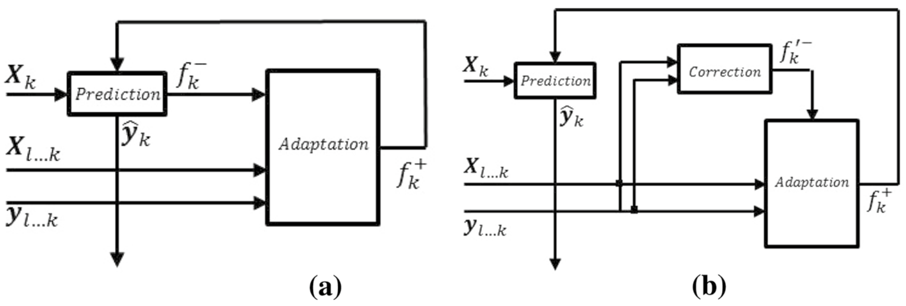


Fig. 2 **a** Adaptation scheme. **b** Adaptation scheme with retrospective correction. Here $1 \leq l \leq k$ and $f_k^{\prime -}$ represents the result of retrospective correction. Depending on the algorithm, inputs can be optional.

$$g_k(\boldsymbol{X}_k, \boldsymbol{y}_k, \Theta_g, f_k^-, \hat{\boldsymbol{y}}_k) : f_k^- \rightarrow f_k^+. \tag{3}$$

or alternatively as $f_k^+ = f_k^- \circ g_k$ for conciseness. Note f_k^- and $\hat{\boldsymbol{y}}_k$ are optional arguments and Θ_g is the set of parameters of g. The function is propagated into the next batch as $f_{k+1}^- = f_k^+$ and predictions themselves are always made using the a priori function f_k^-.

We examine a situation when a choice of multiple, different AMs, $\{\emptyset, g_1, ..., g_H\} = G$, is available. Any AM $g_{h_k} \subset G$ can be deployed on each batch, where h_k denotes the AM deployed at batch k. As the history of all adaptations up to the current batch, k, have in essence created f_k^-, we call that sequence $g_{h_1}, ..., g_{h_k}$ an *adaptation sequence*. Note that we also include the option of applying no adaptation denoted by $\emptyset$. In this formulation, only one element of G is applied for each batch of data. Deploying multiple adaptation mechanisms on the same batch are accounted for with their own symbol in G. Figure 2a illustrates our initial formulation of adaptation.

3.2 Automated adaptation strategies

In this section we present different generic automated adaptive strategies offering flexible deployment of AMs, which can be applied to any adaptive algorithm.

At every batch k, an AM g_{h_k} must be chosen to deploy on the current batch of data. To obtain a benchmark performance, an adaptation strategy which minimizes the error over the incoming data batch $\boldsymbol{X}_{k+1}, \boldsymbol{y}_{k+1}$:

$$f_{k+1}^- = f_k^- \circ g_{h_k}, \quad h_k = \underset{h_k \in 1 \cdots H}{\operatorname{argmin}} \langle (f_k^- \circ g_{h_k})(\boldsymbol{X}_{k+1}), \boldsymbol{y}_{k+1} \rangle \tag{4}$$

where $\langle\rangle$ denotes the chosen error measure, can be used. Since $\boldsymbol{X}_{k+1}, \boldsymbol{y}_{k+1}$ are not yet obtained, this strategy is not applicable in practice. Also note that this may not be the overall optimal strategy which minimizes the error over the whole dataset. We refer to this strategy as Oracle.

Given the inability to conduct the Oracle strategy, below we list some alternatives. The simplest adaptation strategy is applying the same AM to every batch. The scheme of this strategy is given in Fig. 3a. Note that this scheme fits the “Adaptation” box in Fig. 2a. A more common practice (see Sect. 2) is applying multiple or all available adaptive mechanisms. The scheme of this strategy is given in Fig. 3b which again fits the “Adaptation” box in Fig. 2a.

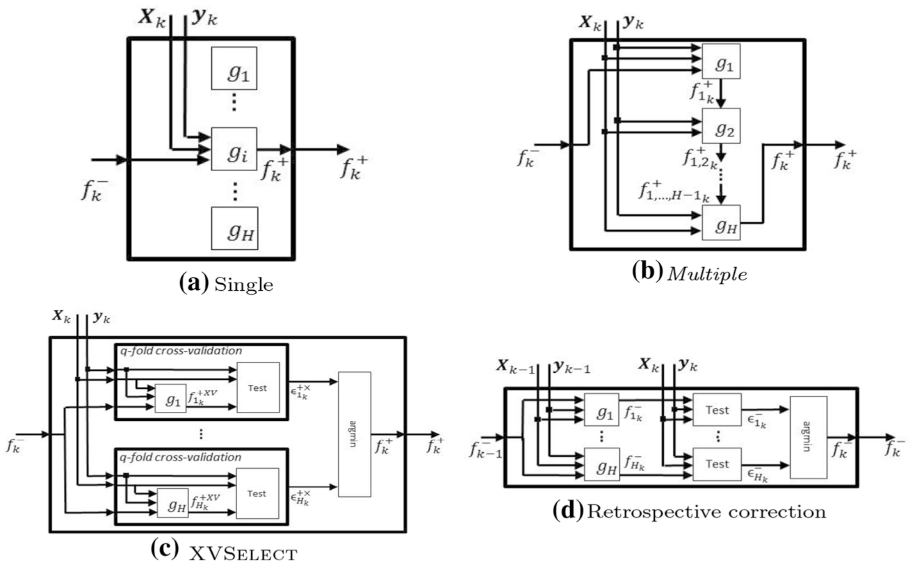


Fig. 3 Automated adaptation strategies

As introduced in Bakirov et al. (2015), it is also possible to use $\mathcal{V}_k$ for the choice of g_{h_k}. Given observations, the a posteriori prediction error $\mathcal{V}_k$ is $\langle (f_k^- \circ g_{h_k})(\boldsymbol{X}_k), \boldsymbol{y}_k \rangle$. However, this is effectively an in-sample error as g_{h_k} is a function of $\{\boldsymbol{X}_k, \boldsymbol{y}_k\}$.[3] To obtain a generalised estimate of the prediction error we apply q-fold[4] cross validation. The cross-validatory adaptation strategy (denoted as XVSelect) uses a subset (fold), $\mathcal{S}$, of $\{\boldsymbol{X}_k, \boldsymbol{y}_k\}$ to adapt; i.e. $f_k^+ = f_k^- \circ g_{h_k}(\{\boldsymbol{X}_k, \boldsymbol{y}_k\}_{\in \mathcal{S}})$ and the remainder, $\not{\mathcal{S}}$, is used to evaluate, i.e. find $\langle f_k^+(\boldsymbol{X}_k)_{\in \not{\mathcal{S}}}, \boldsymbol{y}_{k \in \not{\mathcal{S}}} \rangle$. This is repeated q times resulting in q different error values and the AM, $g_{h_k} \in G$, with the lowest average error measure is chosen. If more than one AM has the same lowest average error, a selection among them is made randomly or utilising prior knowledge. In summary:

$$f_{k+1}^- = f_k^- \circ g_{h_k}, \quad h_k = \underset{h_k \in 1 \cdots H}{\operatorname{argmin}} \langle (f_k^- \circ g_{h_k})(\boldsymbol{X}_k), \boldsymbol{y}_k \rangle^\times \tag{5}$$

where $\langle\rangle^\times$ denotes the cross validated error. The scheme of XVSelect for is given in Fig. 3c.

The next strategy can be used in combination with any of the above strategies as it focuses on the history of the adaptation sequence and retrospectively adapts two steps back. This is called the *retrospective model correction* (Bakirov et al., 2016). Specifically, we set the current model to the output of the AM at batch $k-1$ which would have produced the best estimate in block k:

[3] As a solid example consider the case where f_k^+ is f_k^- retrained using $\{\boldsymbol{X}_k, \boldsymbol{y}_k\}$. In this case $\boldsymbol{y}_k$ are part of the training set and so we risk overfitting the model if we also evaluate the goodness of fit on $\boldsymbol{y}_k$.

[4] In subsequent experiments, $q = 10$

$$f^{-}_{k+1} = f^{-}_{k-1} \circ g_{h_{k-1}} \circ g_{h_k}, \quad h_{k-1} = \underset{h_{k-1} \in 1 \cdots H}{\operatorname{argmin}} \langle (f^{-}_{k-1} \circ g_{h_{k-1}})(\boldsymbol{X}_k), \boldsymbol{y}_k \rangle \tag{6}$$

The potential draws can be again resolved randomly or using prior knowledge. Using the cross-validated error measure in Eq. 6 is not necessary, because $g_{h_{k-1}}$ is independent of $\boldsymbol{y}_k$. Also note the presence of g_{h_k}; retrospective correction does not in itself produce a f_{k+1} and so cannot be used for prediction unless it is combined with another strategy (g_{h_k}). This strategy can be extended to consider the sequence of r AMs while choosing the optimal state for the current batch, which we call r-step retrospective correction:

$$\begin{aligned} f^{-}_{k+1} &= f^{-}_{k-r} \circ g_{h_{k-r}} \circ \cdots \circ g_{h_{k-1}} \circ g_{h_k}, \{h_{k-r} \cdots h_{k-1}\} \\ &= \underset{h_{k-r} \cdots h_{k-1} \in 1 \cdots H}{\operatorname{argmin}} \langle (f^{-}_{k-r} \circ g_{h_{k-r}} \circ \cdots \circ g_{h_{k-1}})(\boldsymbol{X}_k), \boldsymbol{y}_k \rangle \end{aligned} \tag{7}$$

The scheme for *retrospective correction* is given in Fig. 3d. Since the retrospective correction can be deployed alongside any adaptation scheme, we modify the general adaptation scheme (Fig. 2a) accordingly, resulting in Fig. 2b, where Fig. 3d fits in the box "Correction". Notice that when using this approach, the prediction function $f_k(x)$, which is used to generate predictions, can be different from the *propagation* function $f^{'}_k(x)$ which is used as input for adaptation.

An important technical detail for both cross-validatory selection and retrospective correction is the resolution of draws, when two or more AMs show the same predictive performance. The draws appear frequently for classification scenarios with lower batch sizes. In these cases, a prior knowledge on AMs' predictive performance can be used to make a selection.[5] If no such knowledge exists, a random AM, or the AM which minimises the runtime can be chosen.

We next examine the prediction algorithms with respective adaptive mechanisms (the set G) used in this research.

4 Algorithms

For our experiments we have chosen the following algorithms:

- Simple Adaptive Batch Local Ensemble (SABLE) (Bakirov et al., 2015),
- Dynamic Weighted Majority (DWM) (Kolter & Maloof, 2007),
- Paired Learner (PL) (Bach & Maloof, 2010),
- Leveraged Bagging (LB) (Bifet et al., 2010b),
- BLAST (van Rijn et al., 2015).

SABLE is used to address regression problem while the other algorithms address the classification problem. We have developed batch versions of these classification algorithms, which are used in experiments. Our selection of algorithms allows to explore different types of online learning methods and different adaptive mechanisms, and demonstrate that the adaptive strategies described in this paper are in fact generic and can be applied to

[5] This option is used in our experiments.

various adaptive algorithms with multiple AMs. Below the details of model adaptation with each algorithm are presented.

4.1 Simple Adaptive Batch Local Ensemble (SABLE) adaptation

SABLE (Bakirov et al., 2015) uses an ensemble of experts each implemented using a linear model formed through Recursive Partial Least Squares (RPLS) (Joe Qin, 1998). To get the final prediction, the predictions of base learners are combined using input/output space dependent weights (i.e. local learning), which are reflected in the *descriptor* of each expert. SABLE is designed for batch streaming scenario. It supports the creation and merger of base learners.

The SABLE algorithm allows the use of five different adaptive mechanisms (including the possibility of no adaptation). AMs are deployed as soon as the true values for the batch are available and before predicting on the next batch. The six SABLE AMs are described below.[6] It should be noted, that as SABLE was conceived as an experimentation vehicle for AM sequences effects exploration, it does not provide a default custom adaptation strategy.

- **SAM0** (No adaptation). No changes are applied to the predictive model, corresponding to $\emptyset$.
- **SAM1** (Batch learning). The simplest AM augments existing data with the data from the new batch and retrains the model. Given predictions of each expert $f_i \in \mathcal{F}$ on $\boldsymbol{\mathcal{V}}$, $\{\hat{\boldsymbol{y}}_1, ..., \hat{\boldsymbol{y}}_I\}$ and measurements of the actual values, $\boldsymbol{y}$, $\boldsymbol{\mathcal{V}}$ is partitioned into subsets in the following fashion:

$$z = \underset{i \in 1 \cdots I}{argmin} \langle f_i(\boldsymbol{x}_j), y_j \rangle \rightarrow [\boldsymbol{x}_j, y_j] \in \boldsymbol{\mathcal{V}}_z \tag{8}$$

  for every instance $[\boldsymbol{x}_j, y_j] \in \boldsymbol{\mathcal{V}}$. This creates subsets $\boldsymbol{\mathcal{V}}_i, i = 1 \dots I$ such that $\cup_{i=1}^{I} \boldsymbol{\mathcal{V}}_i = \boldsymbol{\mathcal{V}}$. Then each expert is updated using the respective dataset $\boldsymbol{\mathcal{V}}_i$. This process updates experts only with the instances where they achieve the most accurate predictions, thus encouraging the specialisation of experts and ensuring that a single data instance is not used in the training data of multiple experts.
- **SAM2** (Batch learning with forgetting). This AM is similar to one above but uses decay which reduces the weight of the experts historical training data, making the most recent data more important. It is realised via RPLS update with forgetting factor λ. λ is a hyper-parameter of SABLE.
- **SAM3** (Descriptors update / weights change). This AM recalculates the local descriptors using the new batch. This amounts to the change of weights of the experts.
- **SAM4** (Creation of new experts). New expert s_{new} is created from $\boldsymbol{\mathcal{V}}_k$. Then it is checked whether the newly created expert is similar to any existing experts, in which case the older expert is removed and their descriptors are merged. Finally the descriptors of all resulting experts are updated.
- **SAM5**. SAM2 (Batch learning with forgetting) followed by SAM4 (Creation of New Experts).

[6] See Bakirov et al. (2017) for a full description.

4.2 Batch Dynamic Weighted Majority (bDWM) adaptation

bDWM is an extension of DWM (Kolter & Maloof, 2007) designed to operate on batches of data instead of on single instances as in the original algorithm. bDWM is a global experts ensemble. Assume a set of I experts $S = \{s_i, ..., s_I\}$ which produce predictions $\hat{\boldsymbol{y}} = \{\hat{y}_1, ..., \hat{y}_I\}$ where $\hat{y}_i = s_i(\boldsymbol{x})$ with input $\boldsymbol{x}$ and a set of all possible labels $C = \{c_1, ..., c_J\}$. Then for all $i = 1 \cdots I$ and $j = 1 \cdots J$ the matrix A with following elements can be calculated:

$$a_{i,j} = \begin{cases} 1 & \text{if } \; s_i(\boldsymbol{x}) = c_j \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Assuming weights vector $\boldsymbol{w} = \{w_1, ..., w_I\}$ for respective predictors in S, the sum of the weights of predictors which voted for label c_j is $z_j = \sum_{i=1}^{I} w_i a_{i,j}$. The final prediction is[7]:

$$\hat{y} = \underset{c_j}{\operatorname{argmax}}(z_j). \tag{10}$$

An adaptive model based on bDWM starts with a single expert and can be adapted using an arbitrary sequence of 8 possible AMs (including no adaptation) given below.

- **DAM0** (No adaptation). No changes are applied to the predictive model, corresponding to $\emptyset$.
- **DAM1** (Batch learning). After the arrival of the batch $\mathcal{V}_t$ at time t each expert is updated with it.
- **DAM2** (Weights update and experts pruning). Weights of experts are updated using following rule:

$$w_i^{t+1} = w_i^t * e^{u_i^t}. \tag{11}$$

  where w_i^t is the weight of the ith expert at time t, and u_i^t is its accuracy on the batch $\mathcal{V}_t$. The weights of all experts in ensemble are then normalized and the experts with a weight less than a defined threshold η are removed. It should be noted that the choice of factor $e^{u_i^t}$ is inspired by Herbster and Warmuth (1998), although due to different algorithm settings, the theory developed there is not readily applicable to our scenario. Weights update is different to the original DWM, which uses an arbitrary factor $\beta < 1$ to decrease the weights of misclassifying experts.
- **DAM3** (Creation of a new expert). New expert is created from the batch $\mathcal{V}_t$ and is given a weight of 1.
- **DAM4**. DAM2 (Weights update and experts pruning) followed by DAM1 (Batch learning).
- **DAM5**. DAM1 (Batch learning) followed by DAM3 (Creation of a new expert).
- **DAM6**. DAM2 (Weights Update and Experts Pruning) followed by DAM3 (Creation of a new expert).
- **DAM7**. DAM2 (Weights update and experts pruning) followed by DAM1 (Batch learning) followed by DAM3 (Creation of a new expert).

[7] This definition is adapted from Kuncheva (2004).

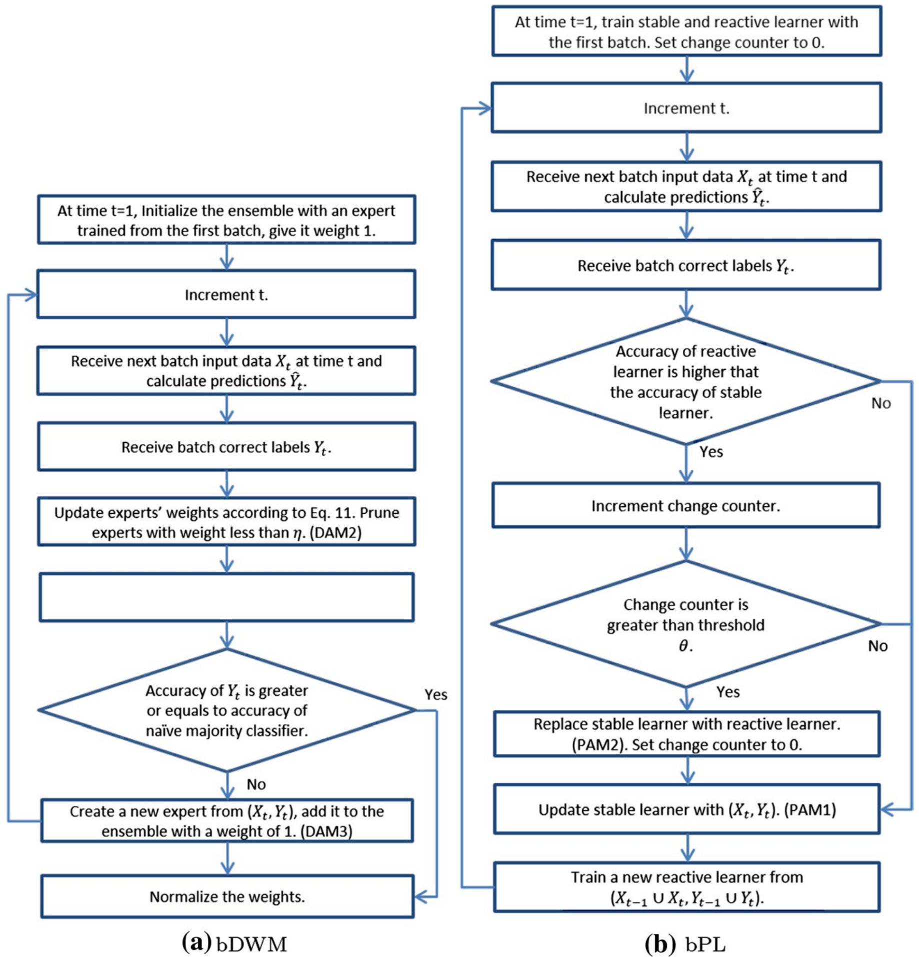

At time t=1, Initialize the ensemble with an expert trained from the first batch, give it weight 1.
Increment t.
Receive next batch input data X_t at time t and calculate predictions $\hat{Y}_t$.
Receive batch correct labels Y_t.
Update experts' weights according to Eq. 11. Prune experts with weight less than η. (DAM2)
Accuracy of Y_t is greater or equals to accuracy of naïve majority classifier.
Yes
No
Create a new expert from (X_t, Y_t), add it to the ensemble with a weight of 1. (DAM3)
Normalize the weights.
(a) bDWM
At time t=1, train stable and reactive learner with the first batch. Set change counter to 0.
Increment t.
Receive next batch input data X_t at time t and calculate predictions $\hat{Y}_t$.
Receive batch correct labels Y_t.
Accuracy of reactive learner is higher that the accuracy of stable learner.
No
Yes
Increment change counter.
Change counter is greater than threshold θ.
No
Yes
Replace stable learner with reactive learner. (PAM2). Set change counter to 0.
Update stable learner with (X_t, Y_t). (PAM1)
Train a new reactive learner from $(X_{t-1} \cup X_t, Y_{t-1} \cup Y_t)$.
(b) bPL

Fig. 4 bDWM and bPL custom adaptation strategies

bDWM (custom adaptive strategy[8]). Having presented the separate adaptive mechanisms, we now describe the bDWM, a batch version of the original DWM. It starts with a single expert with a weight of one. At time t, after an arrival of new batch $\mathcal{V}_t$, experts makes predictions and overall prediction is calculated as shown earlier in this section. After the arrival of true labels all experts learn on the batch $\mathcal{V}_t$ (invoking DAM1), update their weights (DAM2) and ensemble's accuracy u_t is calculated. If u_t accuracy is less than the accuracy of the naive majority classifier (based on all the batches of data seen up to this point) on the last batch, a new expert is created (DAM3). The schematics of this strategy is shown in Fig. 4a. This scheme fits in "Adaptation" boxes in Fig. 2a, b.

[8] To reiterate, we refer to the specific way the AMs are used in original algorithms as "custom adaptive strategy". As the custom adaptive strategy actually defines the algorithm, we will use this term with the name of algorithm (i.e. bDWM) interchangeably.

4.3 Batch Paired Learner (bPL) adaptation

bPL is an extension of PL (Bach & Maloof, 2010) designed to operate on batches of data instead of on single instances as in the original algorithm. bPL maintains two learners—a *stable* learner which is updated with all of incoming data and which is used to make predictions, and a *reactive* learner, which is trained only on the two most recent batches. For this method, three adaptive mechanisms are available, which are described below.

- **PAM0** (No adaptation). No changes are applied to the predictive model, corresponding to $\emptyset$.
- **PAM1** (Updating stable learner). After the arrival of the batch $\mathcal{V}_t$ at time t, stable learner is updated with it.
- **PAM2** (Switching to reactive learner). Current stable learner is discarded and replaced by reactive learner.

bPL (custom adaptive strategy). Having presented the separate adaptive mechanisms, we now describe the bPL, a batch version of the original PL. Its adaptive strategy revolves around comparing the accuracy values of stable (u_s^t) and reactive (u_r^t) learners on each batch of data. Every time when $u_s^t < u_r^t$ a change counter is incremented. If the counter is higher than a defined threshold θ, an existing stable learner is discarded and replaced by the reactive learner, while the counter is set to 0. As before, a new reactive learner is trained from each subsequent batch. The schematics of this strategy are shown in Fig. 4b. This scheme fits in "Adaptation" boxes in Fig. 2a, b.

4.4 Batch Leveraged Bagging (bLB) adaptation

bLB is an extension of LB (Bifet et al., 2010b) designed to operate on batches of data instead of on single instances as in the original algorithm. Leveraged Bagging is based on Online Bagging (Oza & Russell, 2001) algorithm, but includes the improvements such as the removal of experts and addition of new ones based on ADWIN (Bifet & Gavaldà, 2007) change detector, randomization at the ensemble output using output code etc. For this method, four adaptive mechanisms (including no change) are available, which are described below.

- **LAM0** (No adaptation). No changes are applied to the predictive model, corresponding to $\emptyset$.
- **LAM1** (Batch learning). After the arrival of the batch $\mathcal{V}_t$ at time t each expert is updated with it.
- **LAM2** (Removing an existing expert and adding a new one). The expert with the lowest accuracy on the previously seen data is removed, and a new one trained from the most recent batch is added.
- **LAM3**. LAM1 (Batch learning) followed by LAM2 (Removing an existing expert and adding a new one).

bLB (custom adaptive strategy). Having presented the separate adaptive mechanisms, we now describe the bLB, a batch version of the original LB. Its strategy invokes batch learning (LAM1) after the arrival of each batch of data. If ADWIN change detector detects a

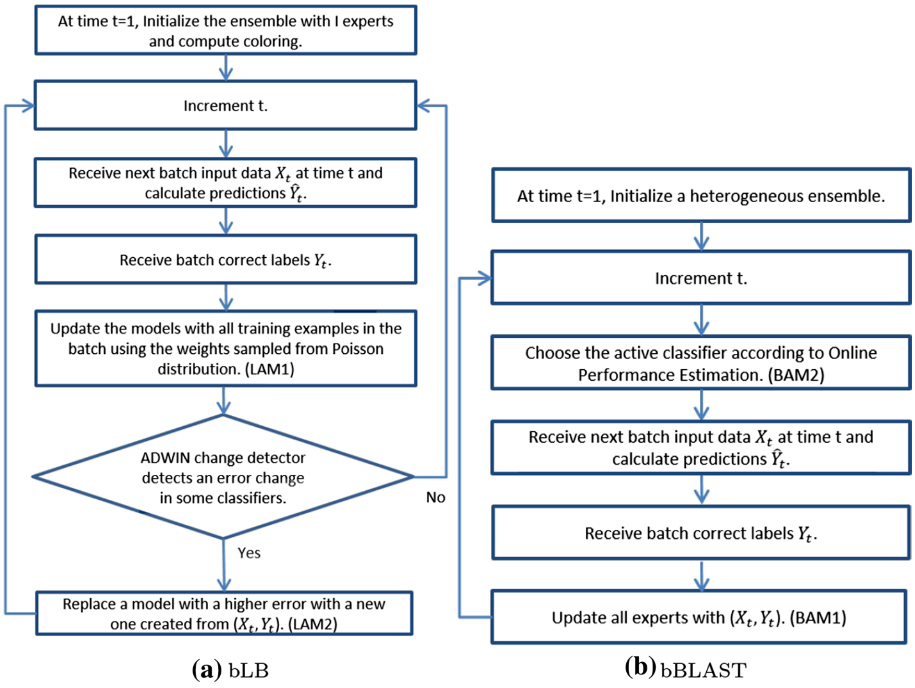

At time t=1, Initialize the ensemble with I experts and compute coloring.
Increment t.
Receive next batch input data X_t at time t and calculate predictions $\hat{Y}_t$.
Receive batch correct labels Y_t.
Update the models with all training examples in the batch using the weights sampled from Poisson distribution. (LAM1)
ADWIN change detector detects an error change in some classifiers.
No
Yes
Replace a model with a higher error with a new one created from (X_t, Y_t). (LAM2)
(a) bLB
At time t=1, Initialize a heterogeneous ensemble.
Increment t.
Choose the active classifier according to Online Performance Estimation. (BAM2)
Receive next batch input data X_t at time t and calculate predictions $\hat{Y}_t$.
Receive batch correct labels Y_t.
Update all experts with (X_t, Y_t). (BAM1)
(b) bBLAST

Fig. 5 bLB and bBLAST custom adaptation strategies

change, the expert with the lowest accuracy on the previously seen data is removed, and a new one trained from the most recent batch is added (LAM2). The schematics of this strategy is shown in Fig. 5a. This scheme fits in “Adaptation” boxes in Fig. 2a, b.

4.5 Batch BLAST (bBLAST) adaptation

bBLAST is an extension of BLAST (van Rijn et al., 2015) designed to operate on batches of data instead of on single instances as in the original algorithm. BLAST is an ensemble method using different types of base learners (as opposed to the ones mentioned above) with Online Performance Estimation for the weighting. For this method, three adaptive mechanisms (including no change) are available, which are described below.

- **BAM0** (No adaptation). No changes are applied to the predictive model, corresponding to $\emptyset$.
- **BAM1** (Batch learning). After the arrival of the batch $\mathcal{V}_t$ at time t each expert is updated with it.
- **BAM2** (Reweighing the experts). For every instance $[\boldsymbol{x}, y] \in \mathcal{V}_t$ experts are reweighed according to Online Performance Estimation.

bBLAST (custom adaptive strategy). Having presented the separate adaptive mechanisms, we now describe the bBLAST, a batch version of the original BLAST. bBLAST invokes the combination of the BAM1 (Batch learning) followed by BAM2 (Reweighing

Table 1 Evaluated adaptive strategies

Result	Description
BestAM	For all of the AMs (e.g. from DAM0 to DAM7 for the bDWM adaptation) we repeatedly deploy the same AM on all of the batches. We then select the best result among all of the runs. Note that this is a post-hoc strategy used for benchmark purposes, as the AM delivering the best result varies from dataset to dataset and is not known in advance
BestAM+RC	The same as BestAM while additionally using retrospective correction after every batch. Note that the best AM here may be different to the one from BestAM
XVSelect	Select AM (i.e. one of AMs from DAM0 to DAM7 for the bDWM adaptation) based on the current data batch using the cross-validatory approach described in Sect. 3.2
XVSelect+RC	The same as XVSelect while additionally using retrospective correction after every batch
Custom	Using custom adaptive strategy
Custom+RC	The same as Custom while additionally using retrospective correction after every batch

the experts) after the arrival of each batch of data. The schematics of this strategy is shown in Fig. 5b. This scheme fits in “Adaptation” boxes in Fig. 2a, b.

5 Experimental results

In the following sub-sections we describe the empirical validation of the proposed approaches. We start by describing the experimental methodology, including experiment settings, specification of datasets, evaluation strategy, libraries and base learners used. We then follow with the comparative analysis of regression and classification results of the proposed and custom adaptive strategies.

5.1 Methodology

The purpose of the experiments[9] in this section was to evaluate the usefulness of the proposed strategies. For this purpose we have performed the empirical comparison of automated adaptation strategies proposed in Sect. 3.2 with custom adaptive strategies and with strategies involving repeated deployment of a single AM. The goal of the automated adaptive strategies is to obtain performance comparable to what one would obtain using a (usually protracted) manually optimised adaptive strategy (including hyper-parameter selection). *Therefore, if the proposed strategies attain comparable, or not significantly worse accuracy levels than the custom strategies, this shall be deemed a success.* This section discusses the results in order of introduced algorithms. For all of the algorithms we compare the MAE/accuracy of strategies listed in Table 1.

For SABLE, the experimentation uses five real world regression datasets listed in Table 5 in “Appendix”. It has been shown, e.g. in Bakirov et al. (2017) and Martin Salvador et al. (2019) that these datasets present different levels of volatility and noise. For the

[9] All of the code except the SABLE algorithm, as well as all the datasets except Oxidizer and Drier can be found on https://github.com/RashidBakirov/multiple-adaptive-mechanisms. SABLE and the specified two datasets could not be shared because of confidentiality reasons.

Table 2 SABLE results

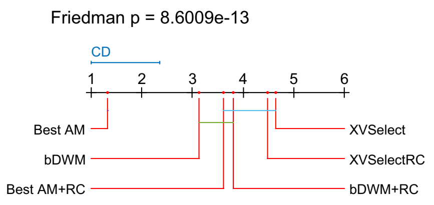
	BestAM	BestAM+RC	XVSelect	XVSelect+RC
$n = 50$				
Catalyst ⇑	0.023 (SAM2)	0.028 (SAM5)	**0.021**	0.023
Oxidiser ⇑	0.490 (SAM2)	0.501 (SAM4)	**0.485**	0.519
Drier ⇑	8.98×10^{-6} (SAM1)	9.78×10^{-6} (SAM0)	9.27×10^{-6}	$\mathbf{6.95\times10^{-6}}$
Debutaniser ↓	**0.117** (SAM1)	0.121 (SAM4)	0.122	0.122
Sulfur ⇓	**0.030** (SAM1)	0.051 (SAM3)	0.060	0.050
$n = 100$				
Catalyst ⇑	0.031 (SAM2)	0.031 (SAM4)	0.030	**0.029**
Oxidiser ⇓	**0.542** (SAM4)	0.559 (SAM4)	0.569	0.566
Drier ⇓	$\mathbf{8.09\times10^{-6}}$ (SAM1)	8.97×10^{-6} (SAM1)	1.20×10^{-5}	1.12×10^{-5}
Debutaniser ⇑	0.117 (SAM1)	0.116 (SAM4)	0.145	**0.112**
Sulfur ⇓	**0.031** (SAM1)	0.058 (SAM2)	0.060	0.054
$n = 200$				
Catalyst ⇑	0.0495 (SAM4)	0.0519 (SAM5)	**0.0492**	0.0495
Oxidiser ↓	0.612 (SAM4)	**0.611** (SAM5)	0.631	0.676
Drier ⇑	5.01×10^{-5} (SAM4)	5.01×10^{-5} (SAM5)	$\mathbf{4.67\times10^{-5}}$	$\mathbf{4.67\times10^{-5}}$
Debutaniser ↑	0.106 (SAM1)	0.105 (SAM4)	**0.104**	0.108
Sulfur ⇓	**0.033** (SAM1)	0.039 (SAM1)	0.049	0.040

The best performance in each row is indicated with bold font. The AM which was found to deliver the best for performance for BestAM and BestAM+RC is indicated in respective columns. Upwards arrow denotes the cases when either XVSelect or XVSelect+RC performs better that BestAM, and downwards arrow denotes the opposite cases. Double lined arrows indicate a significant difference according to Wilcoxon signed-rank test (The Wilcoxon signed-rank test assumes the null distribution is symmetric. This assumption mostly holds for our data) (Wilcoxon, 1945) with $p = 0.05$

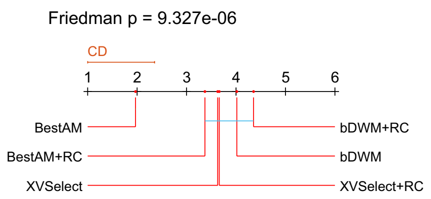
classification algorithms, we use five real world datasets listed in Table 6 and 26 synthetic datasets listed in Table 7 and visualised in Fig. 13 in "Appendix".

For the real world datasets we use prequential evaluation (Dawid, 1984) which is a standard evaluation technique for data streams. For the batch scenario it works as follows; at time t we receive the data batch $\boldsymbol{X}_t$, and predict the values/labels $\hat{\boldsymbol{y}}_t$. Then the true values/labels $\boldsymbol{y}_t$ are made available, and we calculate the error/accuracy of our predictions. Subsequently $\{\boldsymbol{X}_t, \boldsymbol{y}_t\}$ are used for adaptation. Thus, the predictions are always made on unseen data, which is not included in the training data in any form. For synthetic datasets we generate an additional 100 test data instances for each single instance in training data using the same distribution. The predictive accuracy on the batch is then measured on test data relevant to that batch. This test data is not used for training or adapting models.

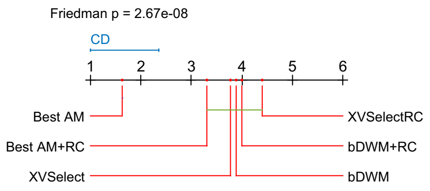
For the classification algorithms, the statistical significance of differences between the results is assessed using the Friedman test with post-hoc Nemenyi test, which are widely used to compare multiple classifiers (Demšar, 2006). The Friedman test checks for statistical difference between the compared classifiers; if so, the Nemenyi test is used to identify which classifiers are significantly better than others. We report the results of the Nemenyi



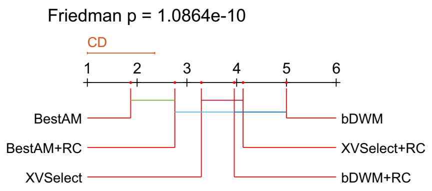
(a) Base learner: NB, $n = 10$



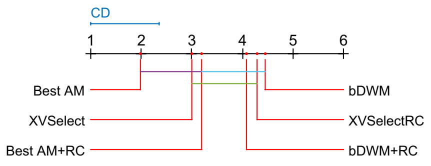
(b) Base learner: HT, $n = 10$



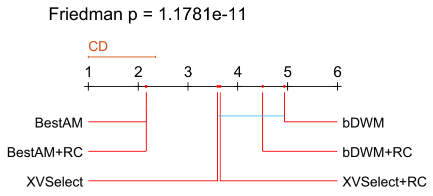
(c) Base learner: NB, $n = 20$



(d) Base learner: HT, $n = 20$



(e) Base learner: NB, $n = 50$



(f) Base learner: HT, $n = 50$

Fig. 6 bDWM adaptation: Nemenyi plots (lower is better) of BestAM, BestAM+RC, XVSelect, XVSelect+RC, Custom (bDWM), Custom+RC (bDWM+RC) strategies for different batch sizes n with NB and HT base learners

tests as Nemenyi plots.[10] They plot the average rank of all methods and the critical difference per batch/base learner. Classifiers that are statistically equivalent are connected by a line.

For bDWM, bPL and bLB, Naive Bayes (NB) and Hoeffding Trees (HT) (Domingos & Hulten, 2000) were used as base learners. Open source libraries Prtools (Duin et al., 2007), Weka (Hall et al., 2009), MOA (Bifet et al., 2010a) and scikit-multiflow (Montiel et al., 2018) were employed. As there is not any randomness involved in the evaluation of datasets, a single run was used to compute the MAE (for regression) and accuracy (for classification) values, except for bLB, where 100 runs were used for each strategy.

5.2 Simple Adaptive Batch Local Ensemble (SABLE) results

Three different batch sizes for each dataset are examined in the simulations together using hyperparameters as tabulated in Table 8 in "Appendix". These parameter combinations were empirically identified using grid search, optimising the performance of the Oracle strategy (Eq. 4).

[10] Freely available code from drawNemenyi (2019) and Cardillo (2009) were used to make these plots.

The results of the experiments using SABLE for batch sizes $n = 50, 100, 200$ are given in Table 2. These results suggest that most of the times XVSelect and XVSelect+RC perform better or comparable to BestAM and BestAM+RC. Overall XVSelect or XVSelect+RC had the lowest MAE with significant difference in 7 experiments out of 15. XVSelect or XVSelect+RC showed comparable (not worse with significant difference) performance to BestAM in 11 experiments. The cases where XVSelect and XVSelect+RC perform noticeably worse are Drier dataset with batch size of 100 and Sulfur dataset with all batch sizes. We relate this to the stability of these datasets. Indeed, the BestAM in all these cases is the slow adapting sequence of SAM1, without any forgetting of the old information. Difference in batch sizes is important for some datasets. This can be related to the frequency of changes and whether they happen within a batch, which can have a negative impact on XVSelect and XVSelect+RC. Retrospective correction (RC) has improved the performance of XVSelect for some cases. For the deployment of single AM, as seen in BestAM and BestAM+RC results, RC is more useful for the larger batch sizes, presumably because more training data prevents overfitting.

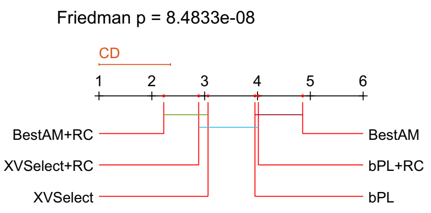
5.3 Batch Dynamic Weighted Majority (bDWM) results

The results of the Nemenyi test are shown in Fig. 6.[11] For four experiments out of six, excluding NB base learner with batch sizes of 10 and 20, XVSelect and XVSelect+RC are both ranked higher than the bDWM (Custom strategy), in some cases significantly so. For batch size 10 with NB as base learner, bDWM performs better than both proposed approaches and for batch size 20, better than XVSelect+RC. The addition of retrospective correction does not seem to bring obvious benefit to adaptive strategies; while improving the results in some experiments, in most of the cases it decreases the accuracy. In terms of batch sizes, increasing n seems to improve the performance of XVSelect with NB base learner. In general, BestAM provides the best results across all experiments, while BestAM+RC performs slightly worse. It may be worth to reiterate that, for all of the classification experiments, the BestAM and BestAM+RC repeatedly deploy the single AM which delivers the best results *specific for particular settings (dataset, batch size, base learner)*. This AM is not known in advance, so this strategy is not attainable in practice and is used for benchmark purposes.

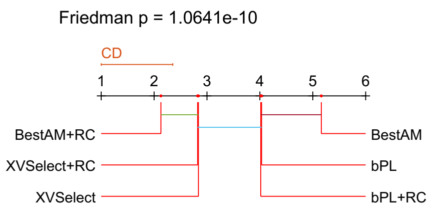
5.4 Batch Paired Learner (bPL) results

For bPL and bPL+RC (Custom and Custom+RC strategies) we have used the threshold of $\theta = 1$ for all the experiments. This value was chosen as it was experimentally established that the lower threshold values tend to provide better results than the higher ones. At the same time, keeping $\theta > 0$ makes use of the change counter mechanism, a characteristic feature of bPL ($\theta = 0$ provided similar results). We present the Nemenyi plots for both base learners on all three batch sizes in Fig. 7. Also for this algorithm, XVSelect and XVSelect+RC show good performance and are ranked higher than the bPL for all batch sizes and base learner combinations. For bPL adaptation, the BestAM+RC performs well for all of the settings, however the performance of BestAM is poor for

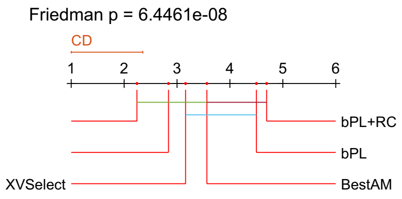
[11] The full results tables with accuracy values of each approach on each dataset are accessible from https://github.com/RashidBakirov/multiple-adaptive-mechanisms/tree/master/results.



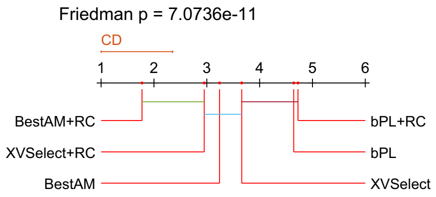
(a) Base learner: NB, $n = 10$



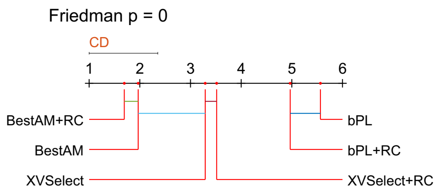
(b) Base learner: HT, $n = 10$



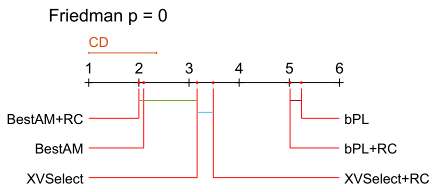
(c) Base learner: NB, $n = 20$



(d) Base learner: HT, $n = 20$



(e) Base learner: NB, $n = 50$



(f) Base learner: HT, $n = 50$

Fig. 7 bPL adaptation: Nemenyi plots (lower is better) of BestAM, BestAM+RC, XVSelect, XVSelect+RC, Custom (bPL), Custom+RC (bPL+RC) strategies for different batch sizes n with NB and HT base learners

the low batch sizes. Retrospective correction appears to be useful for bPL adaptation, providing improvements for BestAM and XVSelect for most settings.

5.5 Batch Leveraged Bagging (bLB) results

bLB adaptation was implemented modifying the existing code from scikit-multiflow. The default hyper-parameters were kept. We present the Nemenyi plots of the average accuracy values of 100 runs for each adaptive strategy for both base learners on all three batch sizes in Fig. 8. The performance of the proposed XVSelect is consistently better than the bLB (Custom strategy) for all of the settings, mostly significantly so. This is even more apparent for higher batch sizes. Behaviour of RC in this case is noteworthy; XVSelect+RC performs consistently worse than XVSelect although still beats the bLB in all of the settings bar one. On the other hand, bLB with RC (Custom+RC strategy) is always better than the bLB. It is possible that for Leveraged Bagging, combining XVSelect and RC makes the adaptation overfit to the last batch, thus reducing the accuracy. For bLB adaptation, the BestAM outperforms the proposed approaches in most of the settings, however there are no significant differences to the performance of XVSelect.

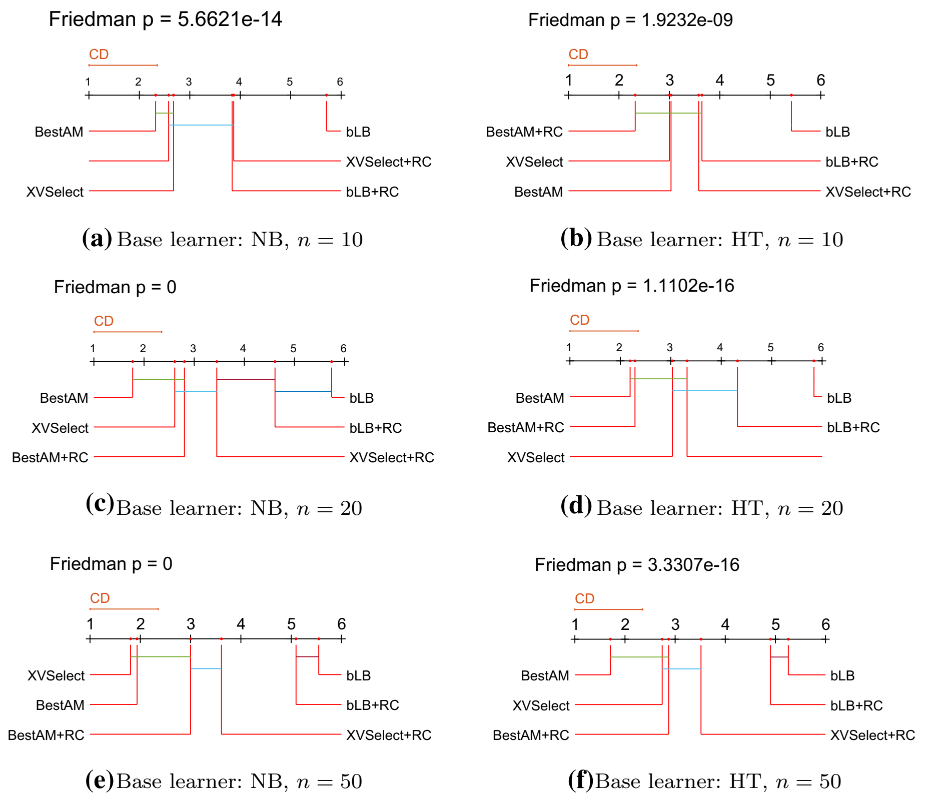
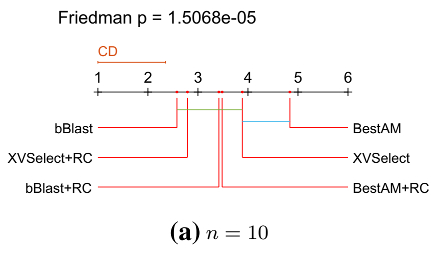
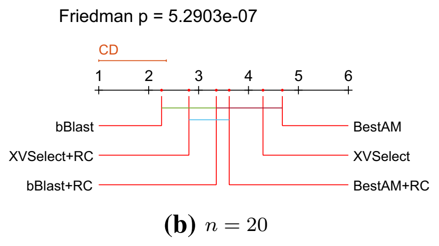


Fig. 8 bLB adaptation: Nemenyi plots (lower is better) of BestAM, BestAM+RC, XVSelect, XVSelect+RC, Custom (bLB), Custom+RC (bLB+RC) strategies for different batch sizes n with NB and HT base learners

5.6 Batch BLAST (bBLAST) results

bBLAST adaptation was implemented modifying the existing MOA code. In contrast to the algorithms in the previous sections, bBLAST uses not single but multiple base learning algorithms; Hoeffding Tree, Naive Bayes, Perceptron, Stochastic Gradient Descent, and k Nearest Neighbour. All of the parameters of the bBLAST, as well as those of base experts are kept at defaults of MOA. We present the Nemenyi plots of the average the accuracy values of the selected adaptive strategies for all three batch sizes in Fig. 9. The performance of the bBLAST (Custom strategy) is consistently better than the proposed adaptive strategies for all of the settings, though not significantly different than XVSelect+RC for batch sizes $n = 10$ and $n = 20$. The RC effect here is the mirror opposite to the bLB; bBLAST with RC (Custom+RC) always performs worse than bBLAST, however XVSelect+RC always performs better than XVSelect. Performance of BestAM and BestAM+RC strategies for this algorithm is markedly worse than for others as they are often outperformed by XVSelect and XVSelect+RC.





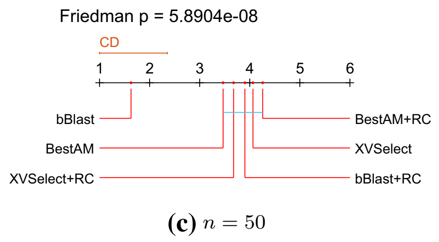
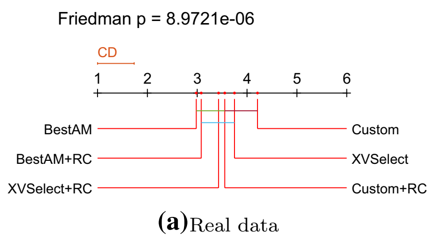


Fig. 9 bBLAST adaptation: Nemenyi plots (lower is better) of BestAM, BestAM+RC, XVSelect, XVSelect+RC, Custom (bBLAST), Custom+RC (bBLAST+RC) adaptive strategies for different batch sizes *n*

Table 3 Comparisons of different approaches

Comparison	Better (significant)	Better	Worse	Worse (significant)
XVSelect versus Custom	11	6	1	3
XVSelect+RC versus Custom	10	6	3	2
XVSelect versus BestAM	2	5	9	5
XVSelect+RC versus BestAM	4	2	3	12
XVSelect+RC versus XVSelect	1	8	11	1
Custom+RC versus Custom	3	9	8	1
BestAM+RC versus BestAM	4	5.5	8.5	3



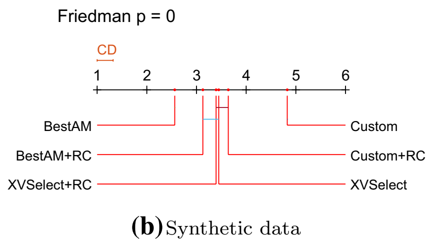


Fig. 10 Nemenyi plots (lower is better) of BestAM, BestAM+RC, XVSelect, XVSelect+RC, Custom, Custom+RC strategies for real and synthetic datasets

Table 4 Relative and absolute (seconds, in brackets) single-core average batch runtimes of XVSelect, XVSelect+RC, Custom, Custom+RC strategies on classification dataset #28 (Power Italy) for different classification algorithms with $n = 50$ and NB base learner

Adaptive Strategy	bPL	bBLAST	bLB	bDWM
Custom	1 (0.036)	1 (0.004)	1 (0.179)	1 (0.056)
XVSelect (2 folds)	2.687 (0.098)	7.67 (0.033)	3.594 (0.644)	112.232 (6.246)
XVSelect (5 folds)	4.896 (0.178)	11.678 (0.05)	8.901 (1.595)	232.83 (12.957)
XVSelect (10 folds)	8.797 (0.32)	17.2 (0.074)	17.622 (3.158)	455.76 (25.363)
Custom+RC	1.003 (0.036)	6.692 (0.029)	3.008 (0.539)	35.105 (1.954)
XVSelect+RC (2 folds)	2.676 (0.097)	13.205 (0.057)	6.062 (1.086)	110.728 (6.162)
XVSelect+RC (5 folds)	5.078 (0.184)	14.912 (0.064)	11.348 (2.033)	224.421 (12.489)
XVSelect+RC (10 folds)	9.047 (0.329)	17.961 (0.077)	20.16 (3.613)	432.138 (24.049)

5.7 Summary of classification results

The conducted experiments give insight on several questions. Firstly, we are interested whether the proposed adaptation strategies XVSelect and XVSelect+RC provide comparable results to the custom strategies or to the best results achieved by a repeated deployment of any AM. Secondly, we would like to know whether the retrospective correction has really a positive effect on the accuracy of the predictions, and if so for which approaches. Finally, we would like to compare the performance of the adaptive strategies on the synthetic data to this on real-world datasets. To answer the first two questions we compare the results from Sects. 5.3–5.6 in Table 3, summing up the number of cases one approach was better and worse than the other across all of the algorithms, batch sizes and base learners (equal performance is represented by 0.5 in both "Better" and "Worse" columns).

In comparison to Custom, XVSelect and XVSelect+RC has better accuracy for most experiments, often with significant difference. For these comparisons XVSelect and XVSelect+RC show similar results. Both XVSelect or XVSelect+RC perform in average worse than BestAM, however, for XVSelect, the performance is comparable (not significantly worse in majority of cases).

Furthermore, we consider the effects of RC separately for each approach, as it has been shown that they could be different. For XVSelect, deploying RC seems to not have a critical effect. The positive effect of RC is more apparent on Custom strategy. For the BestAM it should be noted that the best AM can be different depending on dataset and even for the same dataset it is not necessarily the case that BestAM and BestAM+RC will be based on the same AM. However, we can still say if the best performing AM is known, the deployment of RC is likely to have a negative effect on the accuracy.

Finally, to evaluate the performance of XVSelect and XVSelect+RC on the synthetic vs. real world data, we have compared the results on these datasets separately, across all of the algorithms and settings using Nemenyi plots on Fig. 10. It is possible to observe that the results of the proposed approaches is closer to the BestAM on the real world data, with XVSelect, XVSelect+RC and Custom+RC showing comparable performance. This may be related to the more complicated nature of these datasets, where there may not exist a single AM that markedly optimises the performance, an observation in line with our earlier findings from Bakirov et al. (2015). The performance of XVSelect and XVSelect+RC is comparatively worse on synthetic data, which may be simple enough for a single AM

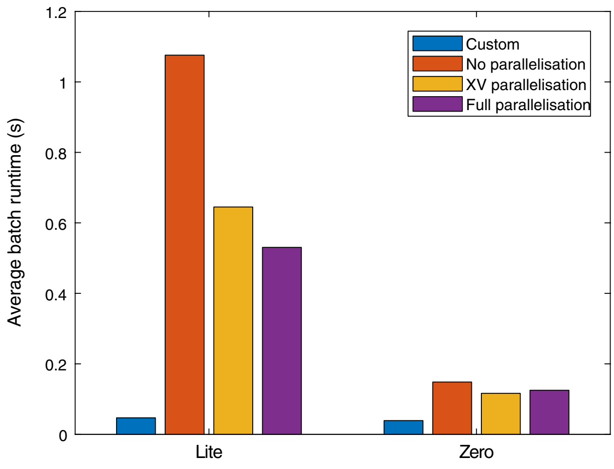


Fig. 11 Average batch runtimes for bDWM Lite and Zero, XVSelect+RC strategy on classification dataset #28 (Power Italy) with $n = 50$ and NB base learner

based adaptive strategy to deliver good results. Even for this case, these two approaches outperform Custom with a significant difference.

5.8 Runtime analysis

We proceed with the analysis of the runtime performance of our approaches. First, we note that with the assumption that the processing time for every batch, including the prediction, adaptation, and accuracy/error calculation is bounded by some constant, which is the case for all of the algorithms we consider, the runtime complexity of any custom adaptive algorithm is $\mathcal{O}(n)$, where n is the number of batches. In this case, the runtime complexity of XVSelect is $\mathcal{O}(|G|qn)$ where $|G|$ is the number of available AMs and q is the number of cross-validation fold, as for every batch every AM with q-fold cross-validation is used. Retrospective correction has the complexity of $\mathcal{O}(|G|n)$, as for every batch every AM is used once. Thus, XVSelect+RC has the complexity of $\mathcal{O}(|G|^2qn)$. Since $|G|$ and q are constants, it follows that $\mathcal{O}(|G|n) \sim \mathcal{O}(|G|qn) \sim \mathcal{O}(|G|^2qn) \sim \mathcal{O}(n)$, hence all the proposed methods are in the same order of runtime complexity as the custom strategies.

For empirical runtime evaluation, we compare the performance of XVSelect, XVSelect+RC, Custom, Custom+RC strategies on classification dataset #28 (Power Italy) for different classification algorithms with $n = 50$ and NB base learner in Table 4,[12] initially without using any parallel processing. This dataset was chosen as it is a relatively large sized real-world dataset. The results show that the performance of our methods vary greatly depending on algorithm; e.g. for XVSelect+RC with 2-fold cross-validation, bPL adaptation has fastest relative average batch processing time (only 2.69 times higher than Custom), whereas bDWM adaptation has the slowest time (110.73 times higher than Custom).

The differences in performances are explainable by the internal characteristics of the algorithms. Batch processing time for XVSelect and XVSelect+RC is proportional to

[12] The results in this section are achieved on quad-core Intel Core i7-7700HQ CPU with core frequency of 2.8 GHz. All adaptive strategies were run 10 times and the average results are reported.

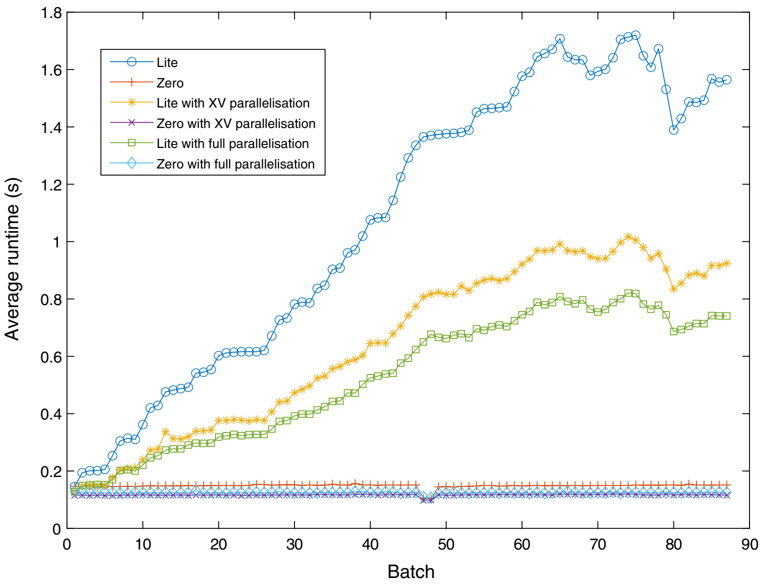


Fig. 12 Average batch runtimes for bDWM Lite and Zero, XVSelect+RC strategy on classification dataset #28 (Power Italy) with $n = 50$ and NB base learner

the batch runtimes with single AMs (e.g. when using Custom strategy). The longer batch runtimes are further extended by the cross-validation and retrospective correction. Therefore, XVSelect and XVSelect+RC for bDWM which has 8 AMs and can have about 20 active experts at the same time, has much higher relative batch runtime than bPL, which has only three AMs and two experts. Other interesting observation is that the RC does not always increase the batch processing time as seen in the example of bPL, which inherently deploys all of the AMs even without RC. This is also the case for bDWM XVSelect and XVSelect+RC, where this may be attributed to the AMs deployed by XVSelect+RC strategy (e.g. less creation of new experts AMs, which notably slow the model down).

Batch processing runtime can be improved by applying parallel processing as both cross-validatory selection and retrospective correction are embarrassingly parallel operations. Fully parallelising the adaptive strategy however requires available $|G|q$ threads which can be prohibitive. Even the fully parallel implementation may not be as efficient as the custom strategy, because the choice of the AM can have an effect on the performance for the subsequent batches. This can be again seen on an example of expert creation AMs.

To illustrate these points a further experiment is undertaken, where two modifications of bDWM are proposed. The first one, bDWM_Lite starts with two experts and includes only two AMs, DAM4 (weights update, experts pruning and batch learning) and DAM7 (weights update, experts pruning, batch learning and expert creation) instead of the original 8, which still allows to run the Custom strategy. bDWM_Lite allows us to test the fully parallel implementation as it requires only 4 threads for this. The second modification, bDWM_Zero, mimics bDWM_Lite, and in addition limits the ensemble to only two experts. This prevents the performance degradation caused by expert creation. We experiment with XVSelect+RC with 2-fold cross-validation and two parallelisation[13] choices, cross-validation (XV) parallelisation where parallel processing is applied to the cross-validation only, and full parallelisation, where in addition to cross-validation, the retrospective correction is also run in parallel. Figure 11 shows the average batch runtimes over the whole dataset. Even without parallelisation, simply reducing the number of AMs from 8 to 2 (bDWM_Lite), results in performance increase by the factor of 6, while parallelisation increases it even further. Limiting the number of experts further reduces the average batch runtime to only 3 times more than the Custom. Note that for bDWM_Zero the parallelisation does not decrease the runtimes by much and that the full parallelisation doesn't outperform XV only parallelisation. This can be attributed to the already reduced runtime due to limited number of experts and the parallel processing overhead which negates increase in performance. Further insights are given in Fig. 12. It can be seen that for bDWM_Lite, average runtime per batch increases as batches come in, due to the increase in experts, however gradually flattens as the number of experts stabilizes around 20. Conversely, for bDWM_Zero, the runtime per batch is stable from the start.

6 Discussion and conclusions

The core aim of this paper was to explore the issue of automating the adaptation of predictive algorithms, which was found to be a rather overlooked direction in otherwise popular area of automated machine learning. In our research, we have addressed this by utilising a simple, yet powerful adaptation framework, which separates adaptation from prediction, defines adaptive mechanisms and adaptive strategies, as well as allows the use of retrospective model correction. This adaptation framework enables the development of generic automated adaptation strategies, which can be deployed on any set of adaptive mechanisms, thus facilitating the automation of predictive algorithms' adaptation.

We have used several automated adaptation strategies, based on cross-validation on the current batch and retrospectively reverting the model to the oracle state after obtaining the most recent batch of data. We postulate that the recently seen data is likely to be more related to the incoming data, therefore these strategies tend to steer the adaptation of the predictive model to achieve better results on the most recent available data.

To confirm our assumptions, we have empirically investigated the merit of automated adaptation strategies XVSelect and XVSelect+RC. For this purpose we have conducted experiments on 10 real and 26 synthetic datasets, exhibiting various levels of adaptation need.

The results are promising, as for the majority of these datasets, the proposed automated approaches were able to demonstrate comparable or better performance to those of specifically designed custom algorithms and the repeated deployment of any single adaptive mechanism. However, it is not the goal of this paper to replace existing custom strategies with the proposed ones. We rather see the benefit of the proposed strategies in their applicability to all algorithms with multiple adaptive mechanisms, so that the designer of the algorithm does not need to spend time and effort to develop a custom adaptive strategy. We have analysed the cases where proposed strategies performed relatively poorly. It is postulated that the reasons for these cases were: (a) lack of change/need for adaptation; (b)

[13] Parallelisation is realised using Matlab Parallel toolbox.

insufficient data in a batch; and (c) relatively simple datasets, all of which have trivial solutions. We have also identified that the choice of algorithm and base learner can affect the performance of proposed strategies.

A benefit of the proposed generic automated adaptation strategies is that they can help designers of machine learning solutions save time by not having to devise a custom adaptive strategy. XVSelect and XVSelect+RC are generally parameter-free, except for the number of cross validation folds, choosing which is trivial.

Naturally, the described strategies come at some cost in performance. This cost varies between different algorithms and is dependent on the number of AMs and other factors, such as number of experts. The runtimes can be reduced by the parallelisation of cross-validatory selection and retrospective correction. It is also conceivable for throughput requirements to be lower for batch learning scenario, as the data is passed to the model only after the whole batch is accumulated.

7 Future work

This research has focused on batch scenario. Adapting the introduced automated adaptive strategies for incremental learning scenario remains a future research question. In that case a lack of batches would for example pose a question of data selection for cross validation. This could be addressed using data windows of static or dynamically changing size. Using an alternative to cross validation can be another solution. Another useful scope of research is focusing on a semi-supervised scenario, where true values or labels are not always available. This is relevant for many applications, amongst them in the process industry.

A dimension which may require more attention is further improvement of the runtime performance of the proposed approaches. An obvious first step in this direction is discarding the less useful, such as “do nothing”, AMs.

Further research directions include theoretical analysis of this direction of research, where relevant expert/bandit strategies may be useful, as well as the experiments with other ML tasks such as time series prediction, clustering and recommender systems. Finally, as we have observed some discrepancies in performance of the proposed approaches across algorithms/datasets/base learners, a natural research direction is to investigate the reasons for these discrepancies. This would also include experimentation with different base learners.

In general, there is a rising tendency of modular systems for construction of machine learning solutions, where adaptive mechanisms are considered as separate entities, along with pre-processing and predictive techniques. One of the features of such systems is easy, and often automated plug-and-play machine learning (Kadlec and Gabrys, 2009; Kedziora et al., 2020). Generic automated adaptive strategies introduced in this paper further contribute towards this automation.

Appendix

See Tables 5, 6, 7, 8 and Fig. 13.

Table 5 Regression datasets with ***N*** instances and ***M*** features

#	Name	*N*	*M*	Description
1	Catalyst activation	5867	12	Highly volatile simulation (real conditions based) of catalyst activation in a multi-tube reactor. Task is the prediction of catalyst activity while inputs are flows, concentrations and temperatures (Strackeljan, 2006)
2	Thermal oxidiser	2820	36	Prediction of NO_x exhaust gas concentration during an industrial process, moderately volatile. Input features include concentrations, flows, pressures and temperatures (Kadlec and Gabrys, 2009)
3	Industrial drier	1219	16	Prediction of residual humidity of the process product, relatively stable. Input features include temperatures, pressures and humidities (Kadlec and Gabrys, 2009)
4	Debutaniser column	2394	7	Prediction of butane concentration at the output of the column. Input features are temperatures, pressures and flows (Fortuna et al., 2005)
5	Sulfur recovery	10,081	6	Prediction of SO_2 in the output of sulfur recovery unit. Input features are gas and air flow measurements (Fortuna et al., 2003)

Table 6 Real world classification datasets with ***N*** instances, ***M*** features and ***C*** classes

#	Name	*N*	*M*	*C*	Brief description
27	Australian electricity prices (Elec2)	27,887	6	2	Widely used concept drift benchmark dataset thought to have seasonal and other changes as well as noise. Task is the prediction of whether electricity price rises or falls while inputs are days of the week, times of the day and electricity demands (Harries, 1999)
28	Power Italy	4489	2	4	The task is prediction of hour of the day (03:00, 10:00, 17:00 and 21:00) based on supplied and transferred power measured in Italy (Zhu, 2010; Chen et al., 2015)
29	Contraceptive	4419	9	3	Contraceptive dataset from UCI repository (Newman et al., 1998) with artificially added drift (Minku et al., 2010)
30	Iris	450	4	4	Iris dataset (Anderson, 1936; Fisher, 1936) with artificially added drift (Minku et al., 2010)
31	Yeast	5928	8	10	Contraceptive dataset from UCI repository (Newman et al., 1998) with artificially added drift (Minku et al., 2010)

Table 7 Synthetic classification datasets used in experiments, with ***N*** instances and ***C*** classes, from Bakirov and Gabrys (2013)

#	Data type	*N*	*C*	Drift	Noise/overlap
1	Hyperplane	600	2	$2 \times 50\%$ rotation	None
2	Hyperplane	600	2	$2 \times 50\%$ rotation	10% uniform noise
3	Hyperplane	600	2	$9 \times 11.11\%$ rotation	None
4	Hyperplane	600	2	$9 \times 11.11\%$ rotation	10% uniform noise
5	Hyperplane	640	2	$15 \times 6.67\%$ rotation	None
6	Hyperplane	640	2	$15 \times 6.67\%$ rotation	10% uniform noise
7	Hyperplane	1500	4	$2 \times 50\%$ rotation	None
8	Hyperplane	1500	4	$2 \times 50\%$ rotation	10% uniform noise
9	Gaussian	1155	2	$4 \times 50\%$ switching	0–50% overlap
10	Gaussian	1155	2	$10 \times 20\%$ switching	0–50% overlap
11	Gaussian	1155	2	$20 \times 10\%$ switching	0–50% overlap
12	Gaussian	2805	2	$4 \times 49.87\%$ passing	0.21–49.97% overlap
13	Gaussian	2805	2	$6 \times 27.34\%$ passing	0.21–49.97% overlap
14	Gaussian	2805	2	$32 \times 9.87\%$ passing	0.21–49.97% overlap
15	Gaussian	945	2	$4 \times 52.05\%$ move	0.04% overlap
16	Gaussian	945	2	$4 \times 52.05\%$ move	10.39% overlap
17	Gaussian	945	2	$8 \times 27.63\%$ move	0.04% overlap
18	Gaussian	945	2	$8 \times 27.63\%$ move	10.39% overlap
19	Gaussian	945	2	$20 \times 11.25\%$ move	0.04% overlap
20	Gaussian	945	2	$20 \times 11.25\%$ move	10.39% overlap
21	Gaussian	1890	4	$4 \times 52.05\%$ move	0.013% overlap
22	Gaussian	1890	4	$4 \times 52.05\%$ move	10.24% overlap
23	Gaussian	1890	4	$8 \times 27.63\%$ move	0.013% overlap
24	Gaussian	1890	4	$8 \times 27.63\%$ move	10.24% overlap
25	Gaussian	1890	4	$20 \times 11.25\%$ move	0.013% overlap
26	Gaussian	1890	4	$20 \times 11.25\%$ move	10.24% overlap

Column "Drift" specifies number of drifts/changes in data, the percentage of change in the decision boundary and its type. All datasets have 2 input features

Table 8 SABLE hyperparameters for different datasets with batch size n, update weights of descriptors δ_0, δ_1, RPLS forgetting factor λ, kernel width for descriptor construction σ, L RPLS latent variables and K batches

Dataset	n	K	δ_0, δ_1	λ	σ	L
Catalyst	50	117	0, 1	0.5	1	12
Catalyst	100	59	0, 1	0.25	1	12
Catalyst	200	30	0, 1	0.5	1	12
Oxidizer	50	47	0.25, 0.75	0.5	1	3
Oxidizer	100	29	0, 1	0.25	0.01	3
Oxidizer	200	15	0, 1	0.25	0.01	3
Drier	50	25	0, 1	0.25	0.01	16
Drier	100	13	0, 1	0.5	0.1	16
Drier	200	7	0, 1	0.25	0.01	16
Debutaniser	50	47	0.25, 0.75	0.5	1	6
Debutaniser	100	23	0.25, 0.75	0.25	1	6
Debutaniser	200	11	0, 1	0.5	1	6
Sulfur	50	201	0.25, 0.75	0.5	1	7
Sulfur	100	100	0, 1	0.5	0.1	7
Sulfur	200	50	0, 1	0.5	0.1	7

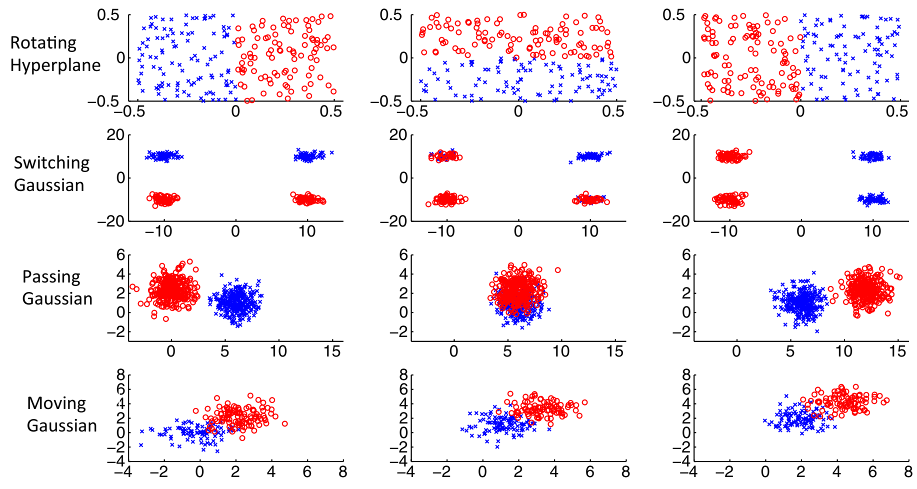


Fig. 13 Synthetic datasets visualisation (Bakirov and Gabrys, 2013)

Acknowledgements We are grateful to anonymous reviewers for their valuable input. We also would like to thank Evonik Industries AG for the provided datasets. Part of the used Matlab code originates from Petr Kadlec and Ratko Grbić. Tomasz Maszczyk has provided helpful coding advice. Thanks to Róman Arango for sharing his statistical wisdom.

References

Alcobé, J. R. (2004). Incremental Hill-Climbing Search Applied to Bayesian Network Structure Learning. In *Proceedings of the 18th European conference on principles and practice of knowledge discovery in databases, Volume 3202 of Lecture notes in computer science*. Springer.

Alippi, C., Boracchi, G., & Roveri, M. (2012). Just-in-time ensemble of classifiers. In *The 2012 international joint conference on neural networks (IJCNN)* (pp 1–8). IEEE.

Anderson, E. (1936). The species problem in iris. *Annals of the Missouri Botanical Garden, 23*(3), 457.

Ba, J., & Frey, B. (2013). Adaptive dropout for training deep neural networks. In *NIPS'13 Proceedings of the 26th international conference on neural information processing systems* (pp. 3084–3092).

Bach, S., & Maloof, M. (2010). A Bayesian approach to concept drift. In *Advances in neural information* (pp. 127–135).

Bakirov, R. (2017). *Multiple adaptive mechanisms for predictive models on streaming data*. Ph.D. thesis, Bournemouth University

Bakirov, R., & Gabrys, B. (2013). Investigation of expert addition criteria for dynamically changing online ensemble classifiers with multiple adaptive mechanisms. In H. Papadopoulos, A. Andreou, L. Iliadis, & I. Maglogiannis (Eds.), *Artificial Intelligence Applications and Innovations* (Vol. 412, pp. 646–656). Berlin: Springer.

Bakirov, R., Gabrys, B., & Fay, D. (2015). On sequences of different adaptive mechanisms in non-stationary regression problems. In *2015 international joint conference on neural networks (IJCNN)* (pp. 1–8).

Bakirov, R., Gabrys, B., & Fay, D. (2016). Augmenting adaptation with retrospective model correction for non-stationary regression problems. In *2016 international joint conference on neural networks (IJCNN)* (pp. 771–779). IEEE.

Bakirov, R., Gabrys, B., & Fay, D. (2017). Multiple adaptive mechanisms for data-driven soft sensors. *Computers & Chemical Engineering, 96,* 42–54.

Bifet, A., & Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. *SIAM International Conference on Data Mining, 7,* 443–448.

Bifet, A., Holmes, G., Gavaldà, R., Pfahringer, B., & Kirkby, R. (2009). New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining—KDD '09* (pp. 139–147).

Bifet, A., Holmes, G., Kirkby, R., & Pfahringer, B. (2010a). MOA: Massive online analysis. *Journal of Machine Learning Research, 11*(52), 1601–1604.

Bifet, A., Holmes, G., & Pfahringer, B. (2010b). Leveraging bagging for evolving data streams. In *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics)*, vol 6321 LNAI (pp. 135–150).

Cardillo, G. (2009). MYFRIEDMAN: Friedman test for non parametric two way ANalysis Of VAriance. Retrieved April 24, 2019, from https://www.mathworks.com/matlabcentral/fileexchange/25882-myfriedman

Carnein, M., Trautmann, H., Bifet, A., & Pfahringer, B. (2020). Towards automated configuration of stream clustering algorithms. In *Communications in computer and information science*, vol 1167 CCIS (pp. 137–143). Springer.

Carpenter, G., Grossberg, S., & Reynolds, J. (1991). ARTMAP: Supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural Networks, 4,* 565–588.

Castillo, G., & Gama, J. (2006). An Adaptive prequential learning framework for Bayesian network classifiers. In J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.) *Knowledge Discovery in Databases: PKDD 2006*. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science (Vol. 4213, pp. 67–78).

Celik, B., & Vanschoren, J. (2021). Adaptation strategies for automated machine learning on evolving data. *Transactions on Pattern Analysis and Machine Intelligence*. https://doi.org/10.1109/TPAMI.2021.3062900

Chen, Y., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A., & Batista, G. (2015). The UCR time series classification archive.

Cinar, A., Parulekar, S. J., Undey, C., & Birol, G. (2003). *Batch fermentation: Modeling: Monitoring, and control*. Boca Raton: CRC Press.

Dawid, A. P. (1984). Present position and potential developments: some personal views: Statistical theory: The prequential approach. *Journal of the Royal Statistical Society Series A (General), 147*(2), 278.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research, 7,* 1–30.

Domingos, P., & Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining—KDD '00* (pp. 71–80).

drawNemenyi. (2019). Retrieved April 24, 2019, from https://github.com/sepehrband/drawNemenyi

Duin, R. P. W., Juszczak, P., Paclik, P., Pekalska, E., de Ridder, D., Tax, D. M. J., & Verzakov, S. (2007). PRTools4.1, A Matlab toolbox for pattern recognition.

Elwell, R., & Polikar, R. (2011). Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks/A Publication of the IEEE Neural Networks Council, 22*(10), 1517–31.

Fern, A., & Givan, R. (2000). *Dynamic feature selection for hardware prediction*. Technical report, Purdue University.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., & Hutter, F. (2015). Efficient and robust automated machine learning. In *Advances in neural information processing systems 28 (NIPS 2015)* (pp. 2962–2970).

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics, 7*(2), 179–188.

Fortuna, L., Rizzo, A., Sinatra, M., & Xibilia, M. (2003). Soft analyzers for a sulfur recovery unit. *Control Engineering Practice, 11*(12), 1491–1500.

Fortuna, L., Graziani, S., & Xibilia, M. (2005). Soft sensors for product quality monitoring in debutanizer distillation columns. *Control Engineering Practice, 13*(4), 499–508.

Friedman, N., & Goldszmidt, M. (1997). Sequential update of Bayesian network structure. In *Proceedings of the Thirteenth conference on Uncertainty in artificial intelligence* (pp. 165–174).

Gabrys, B. (2004). Learning hybrid neuro-fuzzy classifier models from data: To combine or not to combine? *Fuzzy Sets and Systems, 147*(1), 39–56.

Gabrys, B., & Bargiela, A. (1999). Neural networks based decision support in presence of uncertainties. *Journal of Water Resources Planning and Management, 125*(5), 272–280.

Gabrys, B., & Ruta, D. (2006). Genetic algorithms in classifier fusion. *Applied Soft Computing, 6*(4), 337–347.

Gomes Soares, S., & Araújo, R. (2015). An on-line weighted ensemble of regressor models to handle concept drifts. *Engineering Applications of Artificial Intelligence, 37,* 392–406.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter, 11*(1), 10.

Harries, M. (1999). *Splice-2 comparative evaluation: Electricity pricing*. Technical report. The University of South Wales.

Hazan, E., & Seshadhri, C. (2009). Efficient learning algorithms for changing environments. In *ICML'09 Proceedings of the 26th annual international conference on machine learning* (pp. 393–400).

Herbster, M., & Warmuth, M. (1998). Tracking the best expert. *Machine Learning, 29,* 1–29.

Hulten, G., Spencer, L., & Domingos, P. (2001). Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining—KDD '01* (pp. 97–106). ACM Press.

Hutter, F,. Hoos, H. H., & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *LION'05 Proceedings of the 5th international conference on Learning and Intelligent Optimization* (pp. 507–523). Springer.

Ikonomovska, E., Gama, J., & Džeroski, S. (2010). Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery, 23*(1), 128–168.

Jang, J. S. R., Sun, C. T., & Mizutani, E. (1997). *Neuro-fuzzy and soft computing: A computational approach to learning and machine intelligence*. Upper Saddle River: Prentice Hall.

Joe Qin, S. (1998). Recursive PLS algorithms for adaptive data modeling. *Computers & Chemical Engineering, 22*(4–5), 503–514.

Kadlec, P., & Gabrys, B. (2009). Architecture for development of adaptive on-line prediction models. *Memetic Computing, 1*(4), 241–269.

Kadlec, P., & Gabrys, B. (2010). Adaptive on-line prediction soft sensing without historical data. In *The 2010 international joint conference on neural networks (IJCNN)* (pp. 1–8). IEEE.

Kadlec, P., & Gabrys, B. (2011). Local learning-based adaptive soft sensor for catalyst activation prediction. *AIChE Journal, 57*(5), 1288–1301.

Kedziora, D. J., Musial, K., & Gabrys, B. (2012). *Autonoml: Towards an integrated framework for autonomous machine learning*. 2020.12600.

Klinkenberg, R. (2004). Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis, 8*(3), 281–300.

Klinkenberg, R., & Joachims, T. (2000). Detecting concept drift with support vector machines. In *Proceedings of the 7th international conference on machine learning (ICML)* (pp. 487–494).

Kolter, J. Z., & Maloof, M. A. (2007). Dynamic weighted majority: An ensemble method for drifting concepts. *The Journal of Machine Learning Research, 8,* 2755–2790.

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., & Leyton-Brown, K. (2017). Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research, 18*(25), 1–5.

Kuncheva, L. I. (2004). *Combining pattern classifiers: Methods and algorithms*. New York: Wiley-Blackwell.

Lemke, C., & Gabrys, B. (2010). Meta-learning for time series forecasting and forecast combination. *Neurocomputing, 73*(10), 2006–2016.

Lemke, C., Riedel, S., & Gabrys, B. (2009). Dynamic combination of forecasts generated by diversification procedures applied to forecasting of airline cancellations. In *2009 IEEE symposium on computational intelligence for financial engineering* (pp. 85–91). IEEE.

Littlestone, N., & Warmuth, M. (1994). The weighted majority algorithm. *Information and Computation, 108*(2), 212–261.

Lloyd, J. R., Duvenaud, D., Grosse, R., Tenenbaum, J. B., & Ghahramani, Z. (2014). Automatic construction and natural-language description of nonparametric regression models. In *Proceedings of the 28th AAAI conference on artificial intelligence* (pp. 1242–1250). AAAI Press.

Madrid, J. G., Escalante, H. J., Morales, E. F., Tu, W. W., Yu, Y., Sun-Hosoya, L., Guyon, I., & Sebag, M. (2019). Towards AutoML in the presence of Drift: first results. 1907.10772

Martín Salvador, M., Budka, M., & Gabrys, B. (2016). Adapting multicomponent predictive systems using hybrid adaptation strategies with Auto-WEKA in process industry. In *AutoML at ICML 2016, 2011* (pp. 1–8).

Martin Salvador, M., Budka, M., & Gabrys, B. (2019). Automatic composition and optimization of multicomponent predictive systems with an extended auto-WEKA. *IEEE Transactions on Automation Science and Engineering, 16*(2), 946–959.

Minku, L., White, A., & Yao, Xin. (2010). The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering, 22*(5), 730–742.

Mohr, F., Wever, M., & Hüllermeier, E. (2018). ML-Plan: Automated machine learning via hierarchical planning. *Machine Learning, 107*(8–10), 1495–1515.

Montiel, J., Read, J., Bifet, A., & Kegl, B. (2018). Scikit-Multiflow: A multi-output streaming framework. *Journal of Machine Learning Research, 19*(72), 1–5.

Newman, D., Hettich, S., Blake, C., & Merz, C. (1998). UCI repository of machine learning databases.

Nguyen, H., Woon, Y., Ng, W., & Wan, L. (2012). Heterogeneous ensemble for feature drifts in data streams. In *Advances in knowledge discovery and data mining* (pp. 1–12). Springer.

Nguyen, T. D., Maszczyk, T., Musial, K., Zöller, M. A., & Gabrys, B. (2020). Avatar—Machine learning pipeline evaluation using surrogate model. In M. R. Berthold, A. Feelders, & G. Krempl (Eds.), *Advances in Intelligent Data Analysis XVIII* (pp. 352–365). Cham: Springer International Publishing.

Olson, R. S., & Moore, J. H. (2019). *TPOT: A tree-based pipeline optimization tool for automating machine learning* (pp. 151–160). Cham: Springer.

Oza, N. C., & Russell, S. (2001). Online bagging and boosting. *Artificial Intelligence and Statistics, 2001,* 105–112.

van Rijn, J. N., Holmes, G., Pfahringer, B., & Vanschoren, J. (2015). Having a blast: Meta-learning and heterogeneous ensembles for data streams. In: *2015 IEEE international conference on data mining (ICDM)* (pp. 1003–1008). IEEE.

Rossi, A. L. D., de Leon Ferreira, A. C. P., Soares, C., & De Souza, B. F. (2014). MetaStream: A meta-learning based method for periodic algorithm selection in time-changing data. *Neurocomputing, 127,* 52–64.

Ruta, D., Gabrys, B., & Lemke, C. (2011). A Generic Multilevel Architecture for Time Series Prediction. *IEEE Transactions on Knowledge and Data Engineering, 23*(3), 350–359.

Sahel, Z., Bouchachia, A., Gabrys, B., & Rogers, P. (2007). Adaptive mechanisms for classification problems with drifting data. In *Proceeding of the 11th international conference on knowledge-based intelligent engineering systems (KES'2007)* (pp. 419–426). Springer.

Schlimmer, J. C., & Granger, R. H. (1986). Beyond incremental processing: Tracking Concept Drift. In *AAAI-86 Proceedings* (pp. 502–507).

Schmidt, M., & Lipson, H. (2007). Learning noise. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation—GECCO '07* (pp. 1680–1685).

Scholz, M., & Klinkenberg, R. (2007). Boosting classifiers for drifting concepts. *Intelligent Data Analysis, 11*(1), 1–40.

Souza, F., & Araújo, R. (2014). Online mixture of univariate linear regression models for adaptive soft sensors. *IEEE Transactions on Industrial Informatics, 10,* 937–945.

Stanley, K. O. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation, 10*(2), 99–127.

Strackeljan, J. (2006). *NiSIS Competition 2006-Soft Sensor for the adaptive Catalyst Monitoring of a Multi-Tube Reactor*. Technical report, Universität Magdeburg.

Street, W. N., & Kim, Y. S. (2001). A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the 7th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 377–382).

Vakil-Baghmisheh, M. T., & Pavešić, N. (2003). A fast simplified fuzzy ARTMAP network. *Neural Processing Letters, 17*(3), 273–316.

Veloso, B., Gama, J., & Malheiro, B. (2018). Self hyper-parameter tuning for data streams. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. vol 11198 LNAI (pp. 241–255). Springer.

Wang, H., Fan, W., Yu, P. S., & Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on knowledge discovery and data mining—KDD '03* (pp. 226–235). ACM Press.

Wasserman, L. (2000). Bayesian model selection and model averaging. *Journal of Mathematical Psychology, 44*(1), 92–107.

Widmer, G., & Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts. *Machine Learning, 23*(1), 69–101.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin, 1*(6), 80. https://doi.org/10.2307/3001968.

Zhu, X. (2010). Stream Data Mining Repository. Retrieved April 19, 2013, from http://www.cse.fau.edu/~xqzhu/stream.html

Zliobaite, I. (2011). Combining similarity in time and space for training set formation under concept drift. *Intelligent Data Analysis, 15*(4), 589–611.

Zliobaite, I., & Kuncheva, L. I. (2010). Theoretical window size for classification in the presence of sudden concept drift. Technical report, CS-TR-001-2010, Bangor University, UK.